Name: Mostafa Elgebily email adres: mostafaahmedgalalelgebily@student.utwente.nl

Student id: s2097230

Educational Program: Systems & Control

Exercise 2: Image registration and geometrical transforms

The application addressed in this exercise is stitching images of geometrical maps. Figure 1 shows four images of nautical maps. The images partly overlap. The purpose of this exercise is to get a single image which is geometrically correct so that it can be used for navigation. First the images are glued together in a single image, which is then processed to undo the geometrical deformation. We start with image 1. This is the image to which the other images will be glued. In the first parts of this exercise, the image stitching will be done sequentially (one image after another). The end result will then depend on the order in which we process the image. In the last part of this exercise, we consider stitching all images together in one go. That is, finding the geometrical transforms of all three images in a single optimization step. In this latter case, the order in which the images are processed will not influence the end result.

For inspiration, you may want to check the example: `Feature Based Panoramic Image Stitching` which can be found in Matlab's documentation.

Assessment:

Part 1 (successful stitching of 2 images):	max grade: 6
Part 2 (successful stitching of 4 images):	max grade: 8
Part 3 (making the stitched map geometrically correct):	max grade: 9
Part 4 (global optimization):	max grade: 10

Part 4 goes without supervision or explanation except for the one that is given in this document.

image 2

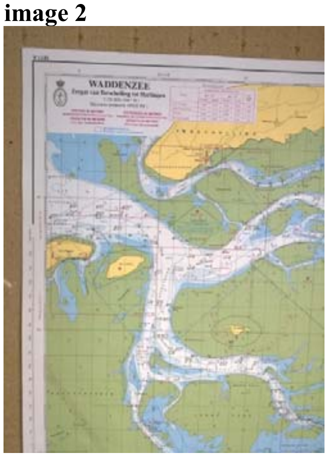


image 1

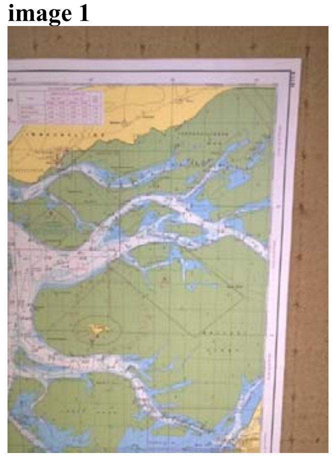


image 3

image 4

Figure 1 Four partly overlapping images of a nautical map

Software:

The following functions and classes from the ***Image Processing Toolbox*** and the ***Computer Vision System Toolbo***x might be useful:

```
imageSet                        affine2d
cpselect                        projective2d
estimateGeometricTransform      vision.AlphaBlender
imwarp
imref2d
```

Instructions:

Preparation:

1. The Matlab class `imageSets` is a handy tool for management of collection of images. Create a new folder and move the four images to it. Initiate your script by clearing the work space, and by closing all figure windows. Then create an `imageSet` object by the constructor: `images = imageSet('imfolder');` in which `'imfolder'` is the name of the folder which holds the four images. You can now easily access each image by applying the method: `read`. For instance, to access the first image: `im1 = read(images,1);`
2. The Matlab function `estimateGeometricTransform` supports three types of geometrical transforms, i.e. similarity, projective, affine. See the help of the function. The positions and orientations of the camera relative to the map differ in all four images.

   a. What is the appropriate transform type in this application if we want to stitch two images?

   Transform type: projective

   Why? Because the angles in the photos differ (the POV) and so they have to be first projected on the same plane to be stitched together

   b. The transform is specified by a number of parameters. We denote these parameters by `a`, `b`, `c`, etc. Suppose that `(x,y)` are the coordinates of a pixel in the second image. After transformation, they are converted to `(u,v)`. Give the mathematical expression (use matlab syntax) of the two equations that transform `(x,y)` to `(u,v)`:

   X = a.*x + b.*y + c
   Y = d.*x + e.*y + f
   A = g.*x + h.*y + 1
   u = (a.*x + b.*y + c)./A and v = (d.*x + e.*y + f)./A

   c. How many unknown parameters `a`, `b`, … do we have? 4

   d. In order to find these parameters, we manually select a number of points in the second image and pinpoint the corresponding points in the first image. What is the minimal number of corresponding points that is needed to define this transform?

   2

   e. Why this number?

   because this is a 2D image, 2 points correspond to 4 unknowns because we get x1, y1, x2, y2 for points 1 and 2 respectively. which can then be used to give us the unknowns in our transformation matrix.

   f. Are there any other constraints on the selection of points?

   They must not be on the same lines. i.e. they cannot have the same y-values or x-values. They must be completely different.

3. Since there are 4 overlapping images, 6 image pairs exist: 1–2, 1–3, 1–4, 2–3, 2–4, and 3–4. Hence, we can define 6 sets of corresponding points. These points are denoted by ${}^{CS}\mathbf{p}_m^n(k)$ with $k = 1, 2, \cdots, K$. This reads as follows: ${}^{CS}\mathbf{p}_m^n(k)$ is the k-th point in image n that corresponds to a point in image m, and which is expressed in coordinate system CS. Initially, the points are expressed in the *intrinsic coordinate* system of their own image: $CS = n$. Later, this will change, as we will see. A point ${}^{n}\mathbf{p}_m^n(k)$ from image n corresponds with the point ${}^{m}\mathbf{p}_n^m$.

   An elegant way to store sets of corresponding points in a Matlab array are *cell arrays*, which are arrays that can contain data of different type and of different sizes. Indexing in cell arrays occurs with curly brackets. For instance, indexing a single cell element in a cell array is done with `p{n,m}`. Suppose that `p{n,m}` contains the K xy coordinates in a $K \times 2$ array, then all the x coordinates are addressed by `p{n,m}(:,1)`.

   `cpselect` is an annotation tool in Matlab to manually pinpoint corresponding points in two image. To manually build up a collection of the 6 sets of corresponding points, include the following code in your m-script:

```
for n=1:4
    for m=n+1:4
        [p{n,m},p{m,n}] = cpselect(read(images,n),read(images,m),'Wait',true);
    end
end
```

   The set ${}^{n}\mathbf{p}_m^n(k)$ with $k = 1, 2, \cdots, K$ is then stored in the cell array `p{n,m}`. Execute this code and save the resulting array on file for later reference, e.g. `save psets p`. When this is done, insert the comment symbol `%` before the given statements. Instead load the sets from file: `load psets`.

   a. Which criterions did you use to choose the points in the image?

No 2 different points on the same axis line (not the same x or y value for 2 points) and widely spaced in the chosen area to get the most evenly stitched image (no tight groups)

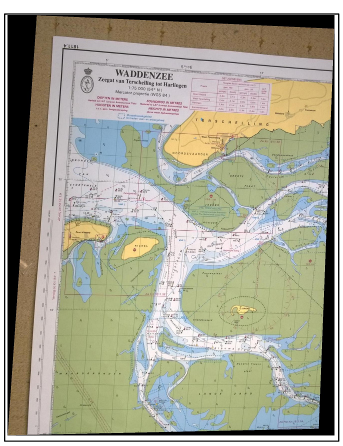
   b. Is the spatial spreading of the markers in the image of importance? If it is important, explain why.

Yes, because the image perspective may change along the image and so you need more varied samples from all around the image that are spaced enough to give a better stitch for different angles and overlaps.

Part I: Stitching image 2 to image 1

4. `p{1,2}` and `p{2,1}` are the sets of corresponding points between image 1 and 2. The function to define the geometrical transform to register image 2 to image 1 is `estimateGeometricTransform`. Use this function to create a so-called `tform` object with the name `tform2`. Use this object to warp image 2 (`imwarp`) towards the domain of image 1. Write the resulting image to file. Inspect the size of the image and compare this with the size image 1.

   Warped image:



Size of image 1:	1632x1224x3 (x3=RGB)
Size of warped image:	1682x1307x3

5. To allow overlaying image 1 with the warped image 2, the two images should have the same image size. As you can see, this is not the case. To elucidate this: there are different coordinate systems at stake:

   - **Pixel indices** are just the row and column indices. These are the subscripts of the 2D array. So, `im(2,15)` returns the grey level at row 2 and column 15. Indices are discrete: always positive integers. The row and column indices are also called the **subscripts** of the array to distinguish them from the linear indices of an array, which happens if you address array elements with just one index, i.e. `im(3)`.
   - **Intrinsic coordinates** are continuously varying coordinates rather than discrete indices. In this coordinate system, locations in an image are positions on a plane, and they are described in terms of x and y (not row and column as in the pixel indexing system). From this perspective, an (x,y) location such as (3.2,5.3) is meaningful, and could be distinct from a pixel with indices (5,3) which would be in intrinsic coordinates (x,y)=(3,5). Note the reversal: in pixel indices, the vertical direction (row) comes first. In intrinsic coordinates, the horizontal direction (x) comes first.
   - **World coordinates** are useful to associate the intrinsic coordinates of one image with the intrinsic coordinates of another image. The world coordinate system associated with an image is an (X,Y) coordinate system that is rotationally aligned with the intrinsic coordinates (x,y). However, the world coordinates may be shifted and scaled. That is: $X = a\,x + x_0$, $Y = b\,y + y_0$.

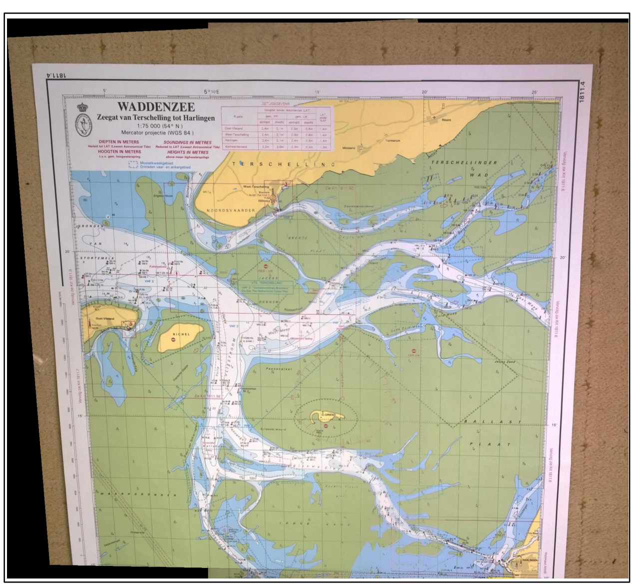
   To overlay image 1 with the warped image 2, both images must share the same world coordinate system. This is currently not yet the case. We need a world coordinate system in which both images can be fitted. This world coordinate system will be denoted by `imref`. which is an object of the class `imref2d`. It will be an unscaled version of the intrinsic coordinates of image 1. To define `imref`, we need to find the spatial limits of the warped image. Spatial limits are the minimum and maximum coordinates in the x and y direction, expressed in im 1 coordinates. We have to compare that with the limits of image 1 since `imref` should hold both images. Calculation of the limits is easily done with `outputLimits` which is a method of the object `tform`. To streamline the process, image 1 and the warped image 2 will be treated in the same way. To do so, the identity transformation, which we will name `tform1`, is needed. Applying this transformation to image 1 yields a warped image 1, which equals image 1 itself (since it is the identity transformation). An identiy operation can be built with the constructor `affine2d` or `projective2d`. Extend your code to build `tform1`.

   Use the `outputLimits` methods of `tform1` and `tform2` to determine for both image 1 and for image 2 the spatial limits. Put the results in arrays `xlims` and `ylims`. Next, use the `min` and the `max` function to determine the needed image size and the `xWorldLimits` and `yWorldLimits` of an `imref2d` object. Create this object, and call it `imref`. See the help of `imref2d`.

Needed image size:	1682x1804
`xWorldLimits:`	[0.5 , 1805.4]
`yWorldLimits:`	[-16.0759 , 1665.9]

   Use the `imwarp` function, the transforms `tform1` and `tform2`, and `imref`, to transform image 1 and image 2. Next overlay image 1 by image 2. For that, you can use the `vision.AlphaBlender` class. A simpler method is shown in the stitching demo (see Blackboard).

   Insert stitched image:



Part II: Sequential stitching of images 2, 3, and 4 to image 1

As before, image 1 will be the reference image. All four images are fully connected to each other. See Figure 2. Therefore, we could repeat the procedure of part I with image 3 and image 4. However, to have a more generally solution we will only use a chain of connected images. See Figure 3. First, image 2 will be stitched to image 1. For that we need to apply a transform that is denoted symbolically by ${}^{1}\mathbf{T}_2$. Next, we stitch image 3 for which ${}^{1}\mathbf{T}_3$ is needed. Finally, we need ${}^{1}\mathbf{T}_4$ for stitching image 4. The four images will be referenced by the variable n. So, what we need are the transforms ${}^{1}\mathbf{T}_n$ with $n=1,\cdots,4$. Note that ${}^{1}\mathbf{T}_1$ is the identity transform.

Since we only want to use the chain, we can only use the connection between image $n-1$ and image n. Thus, only the sets ${}^{n}\mathbf{p}_{n-1}^{n}$ and ${}^{n-1}\mathbf{p}_{n}^{n-1}$ are available. From these sets, the transforms ${}^{n-1}\mathbf{T}_n$ follow. This enable us to calculate ${}^{1}\mathbf{p}_{n}^{n-1}$. These are the points in image $n-1$ that corresponds with points in image n, but that are expressed in the coordinate system of image 1. For instance, ${}^{1}\mathbf{p}_4^3 = {}^{1}\mathbf{T}_2\,{}^{2}\mathbf{T}_3\,{}^{3}\mathbf{p}_4^3$. Once we have ${}^{1}\mathbf{p}_{n}^{n-1}(k)$, we can calculate ${}^{1}\mathbf{T}_n$.

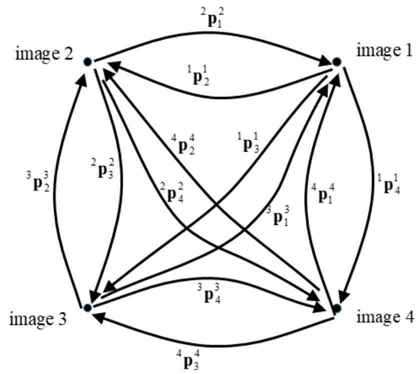


Figure 2. Fully connected

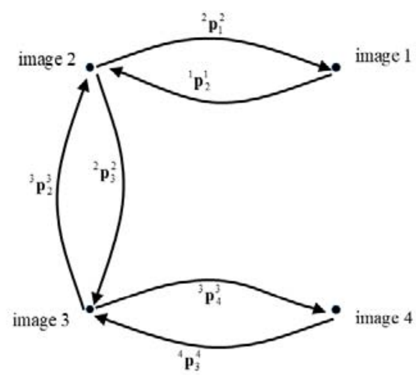
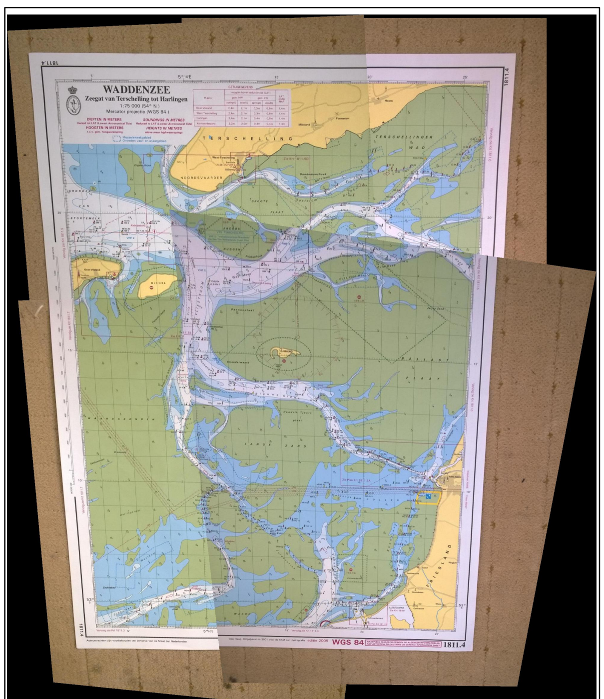


Figure 3. Chained without closed loop

6. Construct the four `tform` objects that corresponds to ${}^{n-1}\mathbf{T}_n$. By definition, ${}^{0}\mathbf{T}_1$ is the identity transform. The other three are the transforms as described above. Put the objects in an array `tform(n)`. Next, the points ${}^{n-1}\mathbf{p}_{n}^{n-1}$ need to be transformed to the domain of image 1, that is, to ${}^{1}\mathbf{p}_{n}^{n-1}$. You can do so with the method `transformPointsForward` which is in the class `tform`. Finally, construct the four `tform` objects that corresponds to ${}^{1}\mathbf{T}_n$. Put them in array `tform1(n)`. Note that ${}^{1}\mathbf{T}_1$ should be the identity transform.
7. We can now stitch the four images by repeating the procedure of part I and using the transforms `tform1(n)`. Since the transforms are available in an array, and the images can be read with `read(images,n)`, this can nicely be done in a for loop. Calculate the outer limits of the four images. Put the results arrays. Then, calculate the minimum and maximum outer limits over the four images. Define a world coordinate system in an `imreg2d` object, warp, stitch, and write to file.

Stitched image:



Size of resulting image:

2407x1975

Discuss the results. Are they according to your expectations? By visual inspection, what can you say about accuracy? What are possible issues?

The 4 images are correctly stitched but the angles of the parallel lines (like the map edges) are not as they should be as well as the extra pieces of the image that stretch out and are unneeded (brown parts). The accuracy of the image is not bad but it cannot be used as a real map seeing that the scales are not representative of real life. Possible issues could be my point selection not being as accurate as needed

8. The sets of corresponding points ${}^{n}\mathbf{p}_{m}^{n-1}$ and ${}^{m}\mathbf{p}_{n}^{m}$ can also be used to assess the accuracy of the stitch. If they are expressed in the coordinate system of image 1, that is ${}^{1}\mathbf{p}_{m}^{n}$ and ${}^{1}\mathbf{p}_{n}^{m}$, they should be the same. Therefore, the difference between these points:

$$e_{n,m}(k) = \left\| {}^{1}\mathbf{p}_{n}^{m}(k) - {}^{1}\mathbf{p}_{m}^{n}(k) \right\| \quad \text{with } k = 1, \cdots, K$$

form a set of error distance. For each image pair, the RMS (root mean square) is:

$$E_{n,m} = \sqrt{\frac{1}{K}\sum_{k=1}^{K} e_{n,m}^{2}(k)}$$

and the overall RMS becomes:

$$RMS = \sqrt{\frac{1}{6}\sum_{n=1}^{4}\sum_{m=n+1}^{4} E_{n,m}^{2}}$$

Calculate the pairwise RMSs and the overall RMS.

$E_{1,2} =$	2.7446	$E_{1,3} =$	15.0083	$E_{1,4} =$	19.2541
$E_{2,3} =$	7.8597	$E_{2,4} =$	20.1864	$E_{3,4} =$	8.0836
$RMS =$	13.7726				

Discuss. What do you observe? Are the results according to expectation? Explain the results.

The results are according to expectations seeing that the error between the images is not very high except for images 2 and 4 where we can see the image being stretched too much to cover for the proportions as well as image 4 not sitting perfectly on top of image 2. However seeing that the errors are low, it is fine. The RMS is also low because the overall image is fairly correct with minimal errors between the overlaps of the 4 sub-images.

Part III: Geometrical rectification

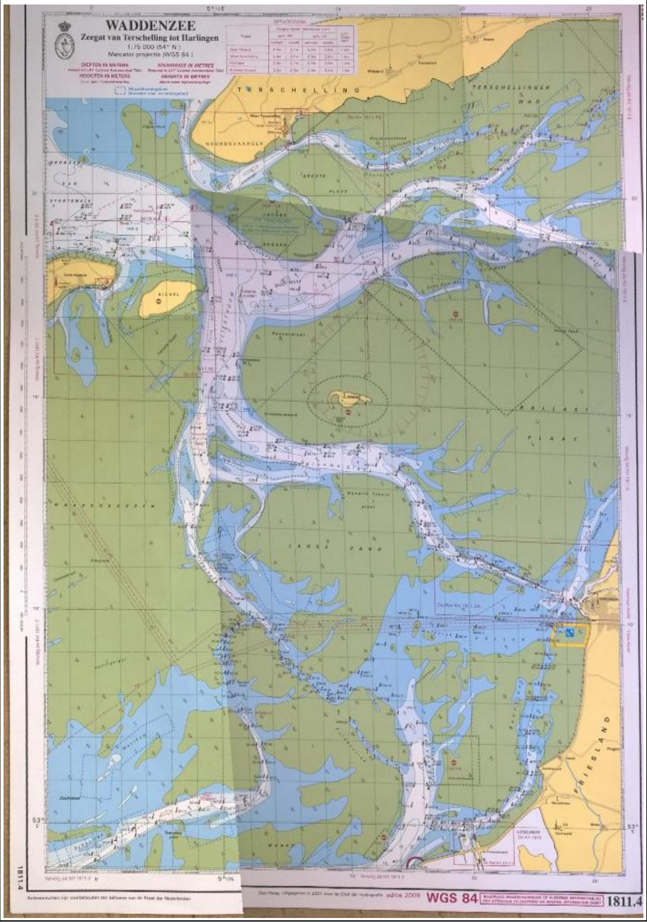
9. The stitched image is not very useful for navigational applications since the map is not a similarity. For instance, directions (angles) in the image are not preserved, and distance ratios in the image are not geometrically correct. In addition, one would want to have the north direction pointing upwards. Apply a geometrical transform to stitched image such that the resulting image is approximately a similarity mapping with the north direction pointing upwards, and angles and distance ratios being preserved. Determine the geometrical transform such that the 100 pixel distance becomes 1 Nm (nautical mile = 1.852 km) in horizontal and vertical direction. You may manually pinpoint (`getpts`) some landmarks in the image for finding the correspondence between geometrical landmarks of the map and the associated points in the image[1]. Use the function `imcrop` to (manually) the irrelevant border of the image.

What are useful landmarks?

The map corners(white paper)
The map ruler on the edges of the paper (use corners again)

[1] Beware: in nautical maps, the unit of a latitudinal (north-south) scale is a degree, in which 1 degree (= 60 minutes) corresponds to 60 Nm. The unit of the longitudinal (west-east) scale is also a degree, but here 1 degree corresponds to 60 cos(latitude) Nm.

Insert the rectified and cropped image:



Part IV: Global optimization

In the previous approach only 3 links (image pairs) from the available 6 were used. As such, the solution was not optimal. To enable global optimization, in which all 6 links are used, a mathematical framework is needed:

- The reference image is image 1. Therefore, we need to find 3 geometrical transforms, ${}^{1}\mathbf{T}_n$ with $n = 2,3,4$. These transforms bring the sets of corresponding points to the domain of image 1, that is ${}^{1}\mathbf{p}_m^n = {}^{1}\mathbf{T}_n {}^{n}\mathbf{p}_m^n$. As explained in question 8, corresponding points in this domain should be the same. With that, an optimization criterion, such as the RMS, can be defined. Optimization boils down to finding the 3 transforms such that the RMS is minimal.
- The 3 transforms ${}^{1}\mathbf{T}_n$ holds internally a number of parameters, say $\alpha_{n,1}$, $\alpha_{n,2}$, … The number of parameters depends on the transformation type (affine, projective, ..). We stack the parameters of each transform into a parameter vector $\boldsymbol{\alpha}_n = \begin{bmatrix} \alpha_{n,1} & \alpha_{n,2} & \cdots \end{bmatrix}^T$, and we write ${}^{1}\mathbf{T}_n = {}^{1}\mathbf{T}(\boldsymbol{\alpha}_n)$ to make clear that ${}^{1}\mathbf{T}_n$ depends on $\boldsymbol{\alpha}_n$.
- The RMS depends on the 3 transforms, and therefore on the vectors $\boldsymbol{\alpha}_n$. To emphasize this, we stack the 3 vectors $\boldsymbol{\alpha}_n$ into a single vector $\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\alpha}_2^{\mathrm{T}} & \boldsymbol{\alpha}_3^{\mathrm{T}} & \boldsymbol{\alpha}_4^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$, and write $RMS(\boldsymbol{\beta})$.
- The optimal solution is represented by the $\boldsymbol{\beta}$ that minimizes $RMS(\boldsymbol{\beta})$. Mathematically, this is expressed as:

$$\boldsymbol{\beta}_{opt} = \arg\min_{\boldsymbol{\beta}} RMS(\boldsymbol{\beta})$$

10. The RMS measures errors are quadratic. What does this mean for pixel pairs with large errors? And with small errors? Is this advantageously? Explain.

large errors (above 1.0) get squared and their magnitude increases while small errors (below 1.0 approx 0.9999) will be squared which will cause their values to decrease drastically. As the error decreases, the squared values decrease even further and become negligible (almost 0)

11. Create a Matlab function, the header of which is defined as:

```
function [RMS,E] = bundle_adjustment(beta, p)
```

`beta` is an array containing the parameters of the 3 geometrical transforms. `p` is the cell arrays that hold the sets of corresponding points that you manually selected in question 3. The function should calculate for each set of corresponding points the value of $E_{n,m}$ and put the results in the array `E`. Here, $E_{n,m}$ is the pairwise RMS as discussed before. `RMS` is the overall root mean square.

A `tform` class stores its parameter in a 3x3 matrix `tform.T`. If it is a `tform` of type `affine2d`, you only need the first two columns: `alpha =tform.T(1:6)`. If it of type `projective2d`, you need the first 8 parameters: `alpha =tform.T(1:6)`. These constructions are useful for packing the parameters into the vector beta. For unpacking this vector, you follow the reverse procedure.

12. Apply the function with as arguments the beta according to the solution of Q6, and calculate the RMS and the $E_{n,m}$, and check whether the answers are the same as in question 8.

$E_{1,2}$ = $E_{1,3}$ = $E_{1,4}$ =

$E_{2,3}$ = $E_{2,4}$ = $E_{3,4}$ =

RMS =

13. Use the matlab optimization function `fminsearch` to find the best `beta.` To use `fminsearch`, you first need to build an *anonymous function*: `f = @(x)bundle_adjustment(x,p);` Initiate the function with the beta from question 12. Calculate the RMS and the $E_{n,m}$, and check whether the answers are the same as in question 8.

$E_{1,2}$ = $E_{1,3}$ = $E_{1,4}$ =

$E_{2,3}$ = $E_{2,4}$ = $E_{3,4}$ =

RMS =

Discuss. What do you observe? Are the results according to expectation? Explain the results.

14. Apply the found geometric transforms, and stitch the images

Discuss. What do you observe? Are the results according to expectation? Explain the results.

Insert the stitched image:

Make sure the m-code fits within the PDF-margin (also the added comments)!

m-code of the function

```
function [RMS,E] = bundle_adjustment(beta,p)
%beta would have tformto1.T(i)
for n=2:4
    for m=n+1:4
        e=0;
        for i=1:4
            e=e+(norm(transformPointsForward(beta(n-1),p{n,m})-...
                transformPointsForward(beta(m),p{m,n})))^2;
            E(n,m)=sqrt(0.25*e);
        end
    end
end

% Getting RMS:
%initialise RMS = 0 (counter)
RMS=0;
for n=1:4
    for m=n+1:4
        RMS=sqrt((1/6)*sum(sum(E.^2)));
    end
end
```

m-code of the main script:

```
clear variables, clear clc, close all
%% Question 1
images = imageSet('imfolder');
im1 = read(images,1);
im2 = read(images,2);
im3 = read(images,3);
im4 = read(images,4);
%% Question 3
% for n = 1:4
%     for m = n+1:4
%         [p{n,m},p{m,n}] = cpselect(read(images,n),read(images,m),'Wait',true);
%     end
% end
% save psets p

load psets.mat
%% Question 4
tform2 = estimateGeometricTransform(p{2,1},p{1,2},'projective');
warp2 = imwarp(im2,tform2);
figure(1)
imshow(warp2);
imwrite(warp2,'warp2.jpg')
%% Question 5
% sz = size(im1);
tform1 = projective2d;
% %gray = im2gray(warp2); %for me to see matrix (size was too big for matlab in RGB)
% [xLimitsOut1,yLimitsOut1] = outputLimits(tform1,[1 sz(2)],[1 sz(1)]);
% [xLimitsOut2,yLimitsOut2] = outputLimits(tform2,[1 sz(2)],[1 sz(1)]);
% % X Limits
% [xlims{1,2}] = xLimitsOut1;
% [xlims{2,1}] = xLimitsOut2;
% % Y limits
% [ylims{1,2}] = yLimitsOut1;
% [ylims{2,1}] = yLimitsOut2;

RA = imref2d(size(im1));
warp1 = imwarp(im1,RA,tform1);
[new_warp2,RB] = imwarp(im2,RA,tform2);
XWorldLimits = [min(min(RA.XWorldLimits,RB.XWorldLimits)),max(max(RA.XWorldLimits,RB.XWorldLimits))];
YWorldLimits = [min(min(RA.YWorldLimits,RB.YWorldLimits)),max(max(RA.YWorldLimits,RB.YWorldLimits))];
imagesize = [fix(YWorldLimits(2)-YWorldLimits(1)) , fix(XWorldLimits(2)-XWorldLimits(1))];
%creating imref
```